CI/CD Pipeline Fundamentals

Terraform-day2 –module-8

Automating Terraform with Jenkins

By the end of this module, you will be able to:

- Explain why manual Terraform execution is risky in real projects
- Describe what CI/CD means in the context of Infrastructure
- Explain how Jenkins pipelines integrate with Terraform
- Read and interpret a basic Jenkinsfile
- Understand how Git branches control Terraform deployments
- Explain why GitOps is important for infrastructure teams

Why Automate Terraform?

The Problem with Manual Workflow

When engineers run Terraform manually:

- Write Terraform code locally
- Run terraform plan on their laptop
- Run terraform apply manually
- Push code to Git later

Why Automate Terraform?

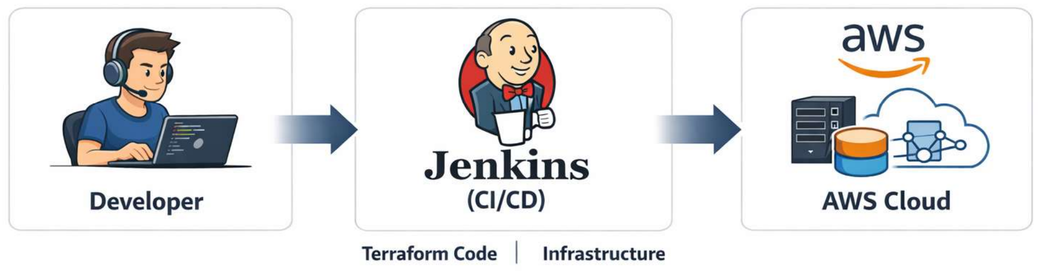
What can go wrong

- Different results on different machines (version mismatch)
- No clear record of who changed what and why
- Validation steps can be skipped
- High risk of applying to the wrong environment
- No formal approval for production changes
- Hard to debug failures in a team

The Solution: CI/CD Pipelines

Benefits (Stronger & more professional):

- Consistent execution every time
- Full audit trail of all infrastructure changes
- Built-in safety via approval gates
- Faster feedback when errors occur
- Reduced human mistakes



What is GitOps?

GitOps means:
👉 **Git is the single source of truth for infrastructure**

Four principles:

- All infrastructure must be in Git (no manual changes)
- Git history = infrastructure change history
- Changes happen only via Pull Requests
- Automation (Jenkins) applies what is in Git

Big benefit:

- To see what is deployed → check Git
- To rollback → revert a commit

What is GitOps?

GitOps means:
👉 **Git is the single source of truth for infrastructure**

Four principles:

- All infrastructure must be in Git (no manual changes)
- Git history = infrastructure change history
- Changes happen only via Pull Requests
- Automation (Jenkins) applies what is in Git

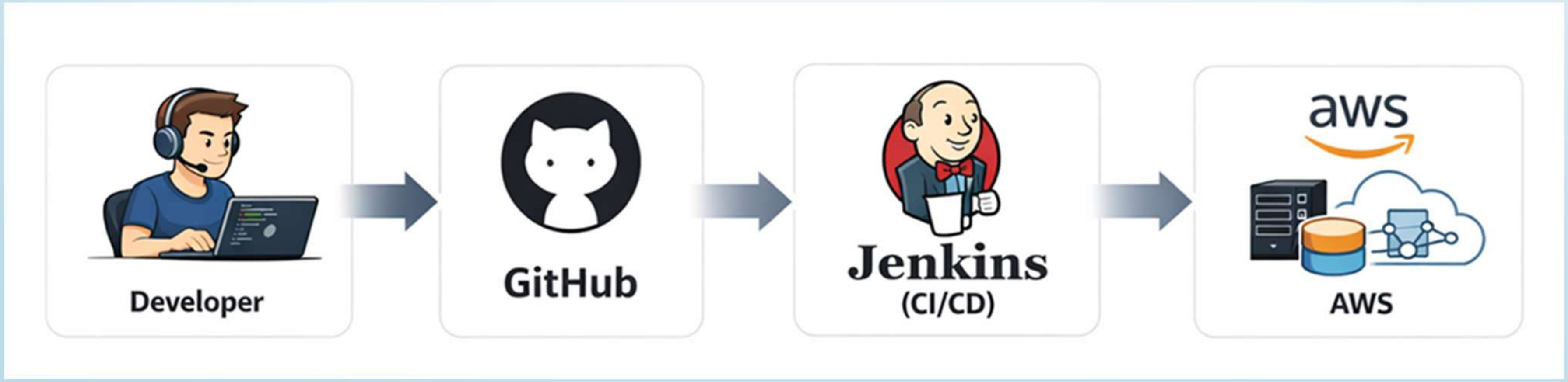
Big benefit:

- To see what is deployed → check Git
- To rollback → revert a commit

Pipeline Architecture

How the Pieces Fit Together (Clearer flow)

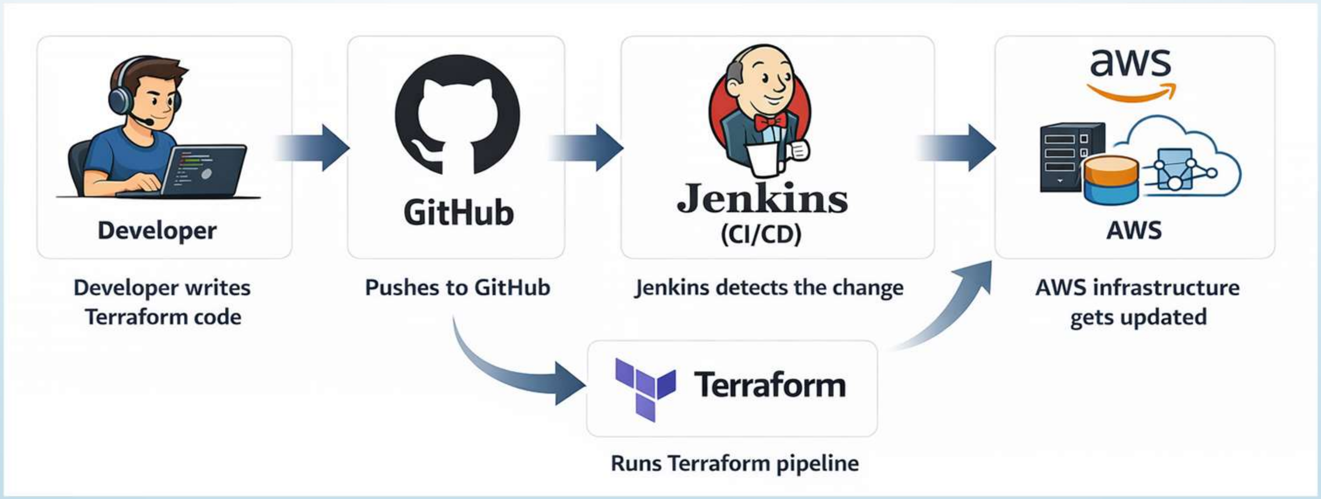
- Developer writes Terraform code
- Pushes to GitHub
- Jenkins detects the change
- Jenkins runs Terraform pipeline
- AWS infrastructure gets updated



Pipeline Architecture

How the Pieces Fit Together (Clearer flow)

- Developer writes Terraform code
- Pushes to GitHub
- Jenkins detects the change
- Jenkins runs Terraform pipeline
- AWS infrastructure gets updated


Developer
Developer writes Terraform code
GitHub
Pushes to GitHub
Jenkins (CI/CD)
Jenkins detects the change
aws
AWS
AWS infrastructure gets updated
Terraform
Runs Terraform pipeline

The Jenkinsfile

A **Jenkinsfile** is a text file that defines your pipeline.
It lives in your Git repository.
Think of it as a **recipe for automation**.

Basic Structure

```
pipeline {
 agent any
 stages {
  stage('Stage Name') {
   steps {
    sh 'command'
   }
  }
 }
}
```

The Jenkinsfile

Stage	**Purpose**	**Terraform Command**
Checkout	Get latest code from Git	Automatic
Init	Download providers	`terraform init`
Validate	Check syntax	`terraform validate`
Plan	Show changes	`terraform plan`
Approval	Human confirmation	`input` step
Apply	Create/update infra	`terraform apply`

Simple Jenkinsfile Example

This pipeline **only checks Terraform but does NOT apply changes yet** — safe for testing.

```
pipeline {
 agent any
 stages {
  stage('Checkout') {
   steps {
    checkout scm
   }
  }
  stage('Terraform Init') {
   steps {
    sh 'terraform init'
   }
  }
stage('Terraform Validate') {
   steps {
    sh 'terraform validate'
   }
  }
  stage('Terraform Plan') {
   steps {
    sh 'terraform plan'
   }
  }
 }
}
```

✅ Runs automatically on every push
❌ Does NOT deploy infrastructure yet

Adding Approval and Apply

Approval Stage

```
•  stage('Approval') {
•    when {
•      branch 'develop'
•    }
•    steps {
•      input message: 'Apply changes?', ok:
   'Apply'
•    }
•  }
```

Terraform Apply Stage

```
•  stage('Terraform Apply') {
•    when {
•      branch 'develop'
•    }
•    steps {
•      sh 'terraform apply -auto-approve'
•    }
•  }
```

👉 **Pipeline pauses and waits for a human to approve.**

Branch Workflow

Branch	**What runs**	**Outcome**
feature/*	Plan only	Safe testing
develop	Plan → Approve → Apply	Deploy to staging
main	Plan → Approve → Apply	Deploy to production
Pull Request	Plan only	No deployments